PATENT
Attorney Docket No. YOR920010421US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of	
Aaron KERSHENBAUM et al.	Examiner: KIM, Jung W.
Serial No: 09/854,031	Art Unit: 2132
Filed: May 11, 2001	
For: AUTOMATED PROGRAM RESOURCE IDENTIFICATION AND ASSOCIATION	

APPEAL BRIEF

Board of Patent Appeals and Interferences
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

The Appellant submits this brief pursuant to 37 C.F.R. §1.192 in furtherance of the Notice of Appeal timely filed in this case on June 22, 2005, setting a two-month shortened statutory period of brief filing expiring August 22, 2005.

Please charge Deposit Account 50-0510 the $500 fee for filing this Appeal Brief. No other fee is believed due with this Appeal Brief, however, should another fee be required please charge Deposit Account 50-0510.

Real Party In Interest

The real party in interest is International Business Machines Corporation.

Related Appeals And Interferences

None.

Status of Claims

Claims 1-100 are pending in this application, with claims 1, 15, 29, 32, 44, 47, 50 and 82 being independent claims.

Claims 1-5, 11-19, 25-36, 42-50, 59-62, 70-72 and 82-100 stand rejected under 35 USC §103 as obvious over Nyanchama, "The Role Graph Model and Conflict of Interest" (hereinafter "Nyanchama") in view of Schmidt, "Data Flow Analysis is Model Checking of Abstract Interpretations" (hereinafter "Schmidt").

Claims 6-10, 20-24, 37-41, 52-58, 63-69 and 73-81 stand rejected under 35 USC §103 as obvious over Nyanchama and Schmidt, and in further view of Gong, "Java Security Architecture (JDK 1.2)" (hereinafter "Gong").

Claim 51 stands rejected under 35 USC §103 as obvious over rejected as obvious over Nyanchama and Schmidt, and in further view of U.S. Patent No. 5,428,554 to Laskoski (hereinafter "Laskoski").

Status of Amendments

Claims 7, 21, 50, 60, 61, 71, 81, 82, 87 and 88 were amended after the Final Rejection dated March 22, 2005.

Summary of the Claimed Subject Matter

The present invention deals with identifying a set of permissions used by a computer program. Application, pg. 1, ln. 3-4, pg. 7, ln. 6-7. To achieve this, the computer program is graphed, wherein the graph represents the collection of program code. Application, pg. 8, ln. 1-8. The collection of code includes codes constructed from a group of codes including basic blocks, class methods, classes, collections of classes or any combination of these. Application, pg. 23, ln. 20-21. The program graph can be a call graph, an invocation graph, and/or

other suitable graph known to those skilled in the art. Application, pg. 26, ln. 11-13. In a particular embodiment of the invention, data flow analysis is employed in the construction of the program graph. Application, pg. 8, ln. 9-11.

Once the computer program is graphed, bounded paths in the program graph that lead to an authorization test are identified. Application, pg. 7, ln. 7-8. For example, an embodiment of the invention may search a program written in the Java programming language for code paths leading to the AccessController.checkPermission() authorization test. Application, pg. 9, ln. 25 - pg. 10, ln. 1. Paths are bounded by a start node and an end node. Application, pg. 10, ln. 6-8. Program entry points (such as static void main(String[])) are typically start nodes. Application, pg. 10, ln. 3. A stop node is a node that is an authorization test. Application, pg. 8, ln. 20-25.

The program is searched for resources for which authorization is being performed. Application, pg. 18, ln. 3-5. In a Java implementation, for example, an authorization resource may be a parameter object p passed to the authorization test. Application, pg. 10, ln. 6-8. In other programming languages or systems, the authorization resource may be a specific instruction, such as an operating system or supervisor call, or a set of specific methods or procedure calls. Application, pg. 19, ln. 16-18.

Finally, the identified authorization resources are associated with the located bounded paths. Application, pg. 9, ln. 23-24. By doing so, it is possible, prior to loading the mobile code, to automatically prompt the administrator or end-user to authorize or deny the code access to restricted resources, or determine whether authorization testing will be required. Application, Abstract, ln. 9-11. Thus, the present invention helps eliminate conventional trial-and-error techniques for

determining permissions required to enable code to run in a secure programming environment. Application, pg. 2, ln. 4-14.

Grounds for Rejection to be Reviewed on Appeal

I. Claims 1-5, 11-19, 25-36, 42-50, 59-62, 70-72 and 82-100 are rejected under 35 USC §103 as obvious over Nyanchama in view of Schmidt.

II. Claims 6-10, 20-24, 37-41, 52-58, 63-69 and 73-81 are rejected under 35 USC §103 as obvious over Nyanchama and Schmidt, and in further view of Gong.

III. Claim 51 stands rejected under 35 USC §103 as obvious over rejected as obvious over Nyanchama and Schmidt, and in further view of Laskoski.

Argument

A *prima facie* case for obviousness can only be made if the combined reference documents teach or suggest all the claim limitations. MPEP 2143. Furthermore, there must be some suggestion or motivation to modify the reference or to combine reference teachings. *Id.*

Claim 1 recites, in part, "obtaining a collection of code." In rejecting claim 1, the Examiner argues that Nyanchama teaches "obtaining a system defined by a set of authorization." Final OA, pg. 8, par. 16. It is noted that claim 1 does not claim obtaining a "system", but rather a collection of code. This fact is important because Nyanchama has nothing to do with program code analysis.

Nyanchama appears to describe role-based access control (RBAC) algorithms that assist network administrators or security officers manage access rights for various network users. Nyanchama, pages 1-2. For example, the teachings of Nyanchama can be applied to manage

whether one can access the salary information of employees in an organization or whether users in a given project are assigned to one group. Nyanchama, pages 7-8. Nyanchama defines roles as a named set of privileges. Nyanchama, page 5. Role graphs are defined as acyclic, directed graphs in which the nodes represent the roles in a system and the edges represent hierarchical relationship of users. Nyanchama, page 9. For example, a role graph may include nodes such as a company's president, vice presidents and junior employees. Nyanchama, Fig. 3.

Thus, the Appellant respectfully disagrees with the Examiner that Nyanchama teaches the claim element of obtaining a collection of code.

Claim 1 also recites, "providing a program graph representing said collection of code." The Examiner states that Nyanchama teaches "providing a graph representing the system defined by a set of authorizations," but adds that "Nyanchama does not expressly teach translating a collection of code into a graph for analysis." Final OA, pg. 8, par. 16, pg. 9, par. 17.

The Appellant submits that the reason the Examiner cannot find a teaching in Nyanchama for providing a program graph representing said collection of code is because Nyanchama has no relevance to program code analysis. As discussed above, Nyanchama appears to describe algorithms that assist network administrators or security officers manage access rights for various network users.

In an attempt to find a reference that discusses providing a program graph representing a collection of code, the Examiner turns to Schmidt. Schmidt proposes that data flow analysis and model checking are equivalent methods of computer program study. Schmidt, page 38, Abstract. According to Schmidt, "we demonstrate in simplest possible terms that an iterative d.f.a. [data flow analysis] is model check . . ." Schmidt, page 38, Introduction. (The Appellant takes no issue with Schmidt's analysis in as much

as it broadens claims 11, 12, 25, 26, 42, 43, 50, 64 and 69 under the doctrine of equivalents). Schmidt does not discuss security issues in its analysis.

Thus, the Appellant respectfully submits the Examiner has combined the teachings of two completely unrelated references. Schmidt appears to examine two different conventional methods of computer program analysis techniques (data flow analysis and model checking) and argues that they are nominally equivalent. Schmidt, Abstract. Nyanchama, on the other hand, appears to describe authorizations, using techniques for analyzing authorizations to help a computer administrator identify incorrect security roles. Nyanchama, page 1.

The Examiner appears to argue that because Nyanchama describes an object oriented system, and that since Java is an object oriented language, one of ordinary skill in the art would combine Nyanchama with Schmidt since "[i]mplicit in Schmidt is the teaching of program code into discrete parts as a means of devising analysis of the code." Final OA, page 6.

The Appellant respectfully disagrees with the Examiner's conclusion. The Appellant fails to see how an abstract discussion of concepts important to computer administrator to help identify incorrect security roles has nothing to do with the actual implementation of a computer program. Although Nyanchama mentions privileges and access to objects, there is no discussion about how these privileges are implemented or how the privileges are related to program points. Nyanchama, Section 2.1.

Furthermore, the Examiner asserts that Nyanchama teaches "identifying any authorization resources associated in the graph as nodes." Final OA, pg. 8. The Appellant respectfully submits that this is not what claim 1 recites. Claim 1 states, "identifying any authorization resources of said collection of code." As discussed above, Nyanchama has nothing to do with

program code analysis. Therefore, the Appellant respectfully submits that Nyanchama does not teach the claim element of identifying any authorization resources of a collection of code.

In a similar light, the Examiner asserts that Nyanchama teaches "locating any bounded path within a graph." Final OA, pg. 8. Claim 1, on the other hand, recites, "locating any bounded path within said program graph." Nyanchama does not mention or suggest a program graph and therefore cannot has nothing to do with program code analysis. Therefore, the Appellant respectfully submits that Nyanchama does not teach the claim element of identifying any authorization resources of a collection of code.

For at least these reasons, the Appellant respectfully asserts that the Examiner has not established a *prima facie* case of obviousness for claim 1. The Appellant submits that the rejection of claim 1 is improper and requests that the rejection of claim 1 be reversed by the honorable Board.

Independent claims 15, 29, 32, 44, 47, 50 and 82 contain similar claim elements and limitations discussed above for claim 1. Thus, the Appellant respectfully asserts that the Examiner has not established a *prima facie* case of obviousness for claims 15, 29, 32, 44, 47, 50 and 82 for the preceding reasons above. The Appellant submits that the rejections of claims 15, 29, 32, 44, 47, 50 and 82 are therefore also improper and requests that the rejection of claims 15, 29, 32, 44, 47, 50 and 82 be reversed by the honorable Board.

Dependent claims 2-5, 7-14, 16-19, 21-28, 30-36, 38-43, 45, 46, 48, 49, 51-81 and 83-100 are believed allowable for at least the same reasons as the independent claims they respectively depend on. The Appellant therefore requests that the rejections of claims 2-5, 7-14, 16-19, 21-28, 30-36, 38-43, 45, 46, 48, 49, 51-81 and 83-100 be reversed by the honorable Board.

Conclusion

In view of the foregoing, Appellant submits that the rejections of claims 1-5, 7-19, 21-36 and 38-100 are improper and respectfully requests that the rejections of claims 1-5, 7-19, 21-36 and 38-100 be reversed by the Board.

Respectfully submitted,

Dated: August 22, 2005

Ido Tuchman, Reg. No. 45,924
69-60 108th Street, Ste. 503
Forest Hills, NY 11375
Telephone (718) 544-1110
Facsimile (718) 544-8588

Claims Appendix

Claim 1. A method comprising:
employing a computer for:
obtaining a collection of code;
providing a program graph representing said collection of code;
identifying any authorization resources of said collection of code;
locating any bounded path within said program graph; and
associating said any authorization resource with said any bounded path.

Claim 2. A method as recited in claim 1, wherein said collection of code includes codes obtained from a group of codes including basic blocks, class methods, classes, collections of classes or any combination of these.

Claim 3. A method as recited in claim 1, wherein the step of providing includes constructing said program graph through static analysis techniques.

Claim 4. A method as recited in claim 3, wherein the step of constructing includes employing object code.

Claim 5. A method as recited in claim 1, wherein the step of identifying includes finding at least one authorization point in said program graph.

Claim 7. A method as recited in claim 5, wherein the step of finding the authorization point includes finding a AccessController.checkPermission node in said program graph, and finding a java.security.Permission object passed as an argument to the AccessController.checkPermission method.

Claim 8. A method as recited in claim 5, wherein said authorization point is an instruction invocation.

Claim 9. A method as recited in claim 8, wherein said instruction invocation is used in a particular language for said collection of code.

Claim 10. A method as recited in claim 9, wherein said particular language is C#.

Claim 11. A method as recited in claim 1, wherein the step of identifying includes employing data flow analysis.

Claim 12. A method as recited in claim 11, wherein the step of employing includes generating a data flow from said program graph.

Claim 13. A method as recited in claim 1, wherein the step of

identifying any bounded path includes locating a set of start nodes in said program graph, and locating a stop node in said program graph; and said bounded path includes all nodes within the graph bound by said start nodes and said stop node.

Claim 14. A method as recited in claim 1, wherein the step of associating includes associating and aggregating said any authorization resource with said collection of code.

Claim 15. An apparatus comprising:

computing means having:

means for obtaining a collection of code;

means for providing a program graph representing said collection of code;

means for identifying any authorization resources of said collection of code;

means for locating any bounded path within said program graph; and

means for associating said any authorization resource with said any bounded path.

Claim 16. An apparatus as recited in claim 15, wherein said collection of code includes codes obtained from a group of codes including basic blocks, class methods, classes, collections of classes or any combination of these.

Claim 17. An apparatus as recited in claim 15, wherein the means for providing includes means for constructing said program graph through static analysis techniques.

Claim 18. An apparatus as recited in claim 17, wherein the means for constructing includes employing object code.

Claim 19. An apparatus as recited in claim 15, wherein the means for identifying includes means for finding at least one authorization point in said program graph.

Claim 21. An apparatus as recited in claim 19, wherein the means for finding the authorization point includes finding a AccessController.checkPermission node in said program graph, and finding a java.security.Permission object passed as an argument to the AccessController.checkPermission method.

Claim 22. An apparatus as recited in claim 19, wherein said authorization point is an instruction invocation.

Claim 23. An apparatus as recited in claim 22, wherein said instruction invocation is used in a particular language for said collection of code.

Claim 24. An apparatus as recited in claim 23, wherein said particular language is C#.

Claim 25. (original) An apparatus as recited in claim 15, wherein the means for identifying includes employing data flow analysis.

Claim 26. An apparatus as recited in claim 25, wherein the means for employing includes generating a data flow from said program graph.

Claim 27. An apparatus as recited in claim 15, wherein the means for identifying any bounded path includes locating a set of start nodes in said program graph, and locating a stop node in said program graph; and said bounded path includes all nodes within the graph bound by said start nodes and said stop node.

Claim 28. An apparatus as recited in claim 15, wherein the means for associating includes associating and aggregating said any authorization resource with said collection of code.

Claim 29. A method comprising:

employing a computer including the steps of:

obtaining a collection of code;

providing a program graph representing said collection of code; and

identifying a complete set of authorization resources of said collection of code.

Claim 30. A method as recited in claim 29, if no resource is identified in said step of identifying, further comprising providing an indication that authorization testing is not necessary.

Claim 31. A method as recited in claim 29, further comprising:

identifying any bounded path within said program graph; and

associating any authorization resource identified in the step of identifying with said any bounded path.

Claim 32. An apparatus comprising:

a computer having access to a collection of code, the computer including:

an authorization resource identifier to identify any authorization resources within the collection of code;

a bounded path locator to locate any bounded path within a program graph of said collection of code; and

an associator to associate said any authorization resource with said any bounded path.

Claim 33. An apparatus as recited in claim 32, wherein said collection of code includes codes obtained from a group of codes including basic blocks, class methods, classes, collections of classes or any combination of these.

Claim 34. An apparatus as recited in claim 32, further comprising a program graph constructor to construct said program graph through static analysis techniques.

Claim 35. An apparatus as recited in claim 32, wherein the program graph constructor uses object code of said collection of code.

Claim 36. An apparatus as recited in claim 32, wherein the authorization resource identifier finds at least one authorization point in said program graph.

Claim 38. An apparatus as recited in claim 36, wherein the authorization point includes finding a AccessController.checkPermission node in said program graph, and finding a java.security.Permission object passed as an argument to the AccessController.checkPermission method.

Claim 39. An apparatus as recited in claim 36, wherein said authorization point is an instruction invocation.

Claim 40. An apparatus as recited in claim 39, wherein said instruction invocation is used in a particular language for said collection of code.

Claim 41. An apparatus as recited in claim 40, wherein said particular language is C#.

Claim 42. An apparatus as recited in claim 32, wherein the authorization resource identifier employs data flow analysis.

Claim 43. An apparatus as recited in claim 42, wherein the data flow analysis is generated from said program graph.

Claim 44. An apparatus comprising:

a computing module having access to a collection of code, the computer module including:

an authorization resource identifier to completely identify any authorization resources within a collection of code; and

a bounded path locator to locate any bounded path within a program graph of said collection of code.

Claim 45. An apparatus as recited in claim 44, wherein the authorization resource identifier provides an indication that authorization testing is not necessary if no resource is identified by said authorization resource identifier.

Claim 46. An apparatus as recited in claim 44, further comprising an associator to associate said any authorization resource with said any bounded path.

Claim 47. An apparatus comprising:

a computer including:

means for obtaining a collection of code;

means for providing a program graph representing said collection of code; and

means for identifying a complete set of authorization resources of said collection of code.

Claim 48. An apparatus as recited in claim 47, further comprising means for indicating if authorization testing is necessary.

Claim 49. An apparatus as recited in claim 47, further comprising:

means for identifying any bounded path within said program graph; and

means for associating any authorization resource identified in the step of identifying with said any bounded path.

Claim 50. A method comprising:

employing a computer for:

constructing a program graph from a collection of code using static analysis techniques;

performing a data flow analysis using static analysis techniques;

searching said program graph for any resource for executing said collection of code;

identifying any bounded path within said program graph over which said resource is utilized; and

associating said resource with said collection of code.

Claim 51. The method as recited in claim 50, wherein the step of constructing includes employing source code of said collection of code.

Claim 52. A method as recited in claim 50, wherein the step of constructing includes building an invocation graph of said collection of code

to form said program graph.

Claim 53. A method as recited in claim 50, wherein the step of constructing a program graph includes constructing a call graph of said collection of code to form said program graph.

Claim 54. A method as recited in claim 50, wherein said step of constructing includes using context-sensitivity.

Claim 55. A method as recited in claim 54, wherein said step of using context sensitivity includes using type information for any method receiver and/or any parameter.

Claim 56. A method as recited in claim 55, wherein the step of using type information includes using class and memory allocation site information.

Claim 57. A method as recited in claim 56, wherein the step of using type information includes using per instance information.

Claim 58. A method as recited in claim 57, wherein the step of using instance information includes associating instance information with a node or edge in said program graph.

Claim 59. A method as recited in claim 50, wherein said collection of code includes codes constructed from a group of codes including basic blocks, class methods, classes, collections of classes or any combination of these.

Claim 60. A method as recited in claim 50, wherein the step of searching includes locating a node or edge in said program graph that represents a location where said resource would be utilized.

Claim 61. A method as recited in claim 60, wherein said resource is a resource identifier.

Claim 62. A method as recited in claim 60, wherein said location is an authorization test.

Claim 63. A method as recited in claim 50, wherein said program graph represents an object oriented program.

Claim 64. A method as recited in claim 50, wherein said program graph and said data flow analysis are constructed from Java object code.

Claim 65. A method as recited in claim 61, wherein said resource identifier includes at least one java.security.Permission class.

Claim 66. A method as recited in claim 62, wherein said authorization test is a call to any java.security.AccessController.checkPermission method.

Claim 67. A method as recited in claim 62, wherein said location represents a call to any authorization testing method in any instance of java.lang.SecurityManager and/or one of its subclasses.

Claim 68. A method as recited in claim 55 , wherein said node has a parameter which said type information is a java.security.Permission class.

Claim 69. A method as recited in claim 68, wherein the step of identifying includes locating the constructor for said java.security.Permission class allocation site and using said data flow analysis in identifying any value passed by any parameter to said constructor, wherein the combination of the Java java.security.Permission and a value for any parameter is the used Permission.

Claim 70. A method as recited in claim 50, wherein the step of identifying any bounded path includes locating a set of start nodes in said program graph, and locating a stop node in said program graph; and said bounded path includes all nodes within the graph bound by said start nodes and said stop node.

Claim 71. A method as recited in claim 50, wherein the step of associating includes mapping a subset of said collection of code to said resource.

Claim 72. A method as recited in claim 71, wherein the subset of said collection of code includes nodes in said bounded path.

Claim 73. A method as recited in claim 72, further comprising employing privileged code wherein said stop node represents the method java.security.AccessController.checkPermission(); and

employing said start nodes are any of root nodes in said program graph or a node representing the method java.security.AccessController.doPrivileged().

Claim 74. A method as recited in claim 73, further comprising connecting a used Permission with said privileged ~~Java~~ code.

Claim 75. A method as recited in claim 74, further comprising connecting a used Permission with any node in said program graph prior to said java.security.AccessController.doPrivileged() node.

Claim 76. A method as recited in claim 74, wherein said step of associating includes connecting a used Permission for each java.security.AccessController.checkPermission() in said program graph.

Claim 77. A method as recited in claim 76, wherein said step of associating includes connecting said used Permission from each node in said program graph to each method associated with said program graph.

Claim 78. A method as recited in claim 77, wherein said step of associating includes connecting said used Permission from each method in said program graph to each class in said program graph.

Claim 79. A method as recited in claim 78, wherein said step of associating includes connecting said used Permission from each class in said program graph to collection of classes.

Claim 80. A method as recited in claim 79, wherein said step of associating includes connecting said used Permission is associated from each class in said program graph to collection of classes.

Claim 81. A method as recited in claim 50, further comprising employing said resource in executing said collection of code.

Claim 82. A method comprising:

statically detecting resources for a collection of code written in a computer programming language, by including the steps of:

calculating if any code of said collection of code is part of a program graph;

identifying any resource for said collection of code;

determining any bounded path of nodes within said program graph which constrain said resources of said collection of code; and

associating said any resource with said collection of code within said bounded path of nodes in said program graph.

Claim 83. A method as recited in claim 82, wherein the step of calculating includes using an invocation graph having a set of root nodes for said program graph.

Claim 84. A method as recited in claim 82, wherein the step of calculating includes using a call graph having a set of root nodes for said program graph.

Claim 85. A method as recited in claim 83, further comprising determining whether a basic block, method, class is represented as a node in

said invocation graph.

Claim 86. A method as recited in claim 84, further comprising determining whether a basic block, method, class is represented as a node in said call graph.

Claim 87. A method as recited in claim 83, further comprising identifying a node in said invocation graph that identifies said resources, wherein said resources is one or more resources.

Claim 88. A method as recited in claim 84, further comprising a step identifying a node in said invocation graph that identifies said resources, wherein said resources is one or more resources.

Claim 89. A method as recited in claim 83, wherein nodes in said bounded path of nodes are all nodes in the graph that are bounded between start nodes and a stop node.

Claim 90. A method as recited in claim 84, wherein nodes in said bounded path of nodes are all nodes in the graph that are bounded between a set of start nodes and a stop node.

Claim 91. A method as recited in claim 90, wherein all root nodes in said invocation graph are members of the start nodes.

Claim 92. An article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for causing association, the computer readable program code means in said article of manufacture comprising computer readable program code means for causing a computer to effect the steps of claim 1.

Claim 93. An article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for causing identification, the computer readable program code means in said article of manufacture comprising computer readable program code means for causing a computer to effect the steps of claim 29.

Claim 94. An article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for causing association, the computer readable program code means in said article of manufacture comprising computer readable program code means for causing a computer to effect the steps of claim 50.

Claim 95. An article of manufacture comprising a computer usable medium

having computer readable program code means embodied therein for causing statistical detection, the computer readable program code means in said article of manufacture comprising computer readable program code means for causing a computer to effect the steps of claim 80.

Claim 96. A computer program product comprising a computer usable medium having computer readable program code means embodied therein for causing association, the computer readable program code means in said computer program product comprising computer readable program code means for causing a computer to effect the functions of the elements in claim 15.

Claim 97. A computer program product comprising a computer usable medium having computer readable program code means embodied therein for causing association, the computer readable program code means in said computer program product comprising computer readable program code means for causing a computer to effect the functions of the elements in claim 32.

Claim 98. A computer program product comprising a computer usable medium having computer readable program code means embodied therein for causing identification, the computer readable program code means in said computer program product comprising computer readable program code means for causing a computer to effect the functions of the elements in claim 44.

Claim 99. A computer program product comprising a computer usable medium having computer readable program code means embodied therein for causing identification, the computer readable program code means in said computer program product comprising computer readable program code means for causing a computer to effect the functions of the elements in claim 47.

Claim 100. A program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform method steps for identification, said method steps comprising the steps of claim 1.